CARNEGIE-MELLON UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

SPICE PROJECT

AccUnix - Accent style IPC under UNIX

Robert Sansom

21 August 1984

Spice Document S160

Keywords and index categories: Accent, IPC, Unix

Location of machine-readable file: [x]/usr/spicedoc/manual/spiceprogram/sysprog

Abstract

An explanation of how to use the facilities available under UNIX for doing Accent style IPC. In addition, a descripton of how to use the UNIX network server for communicatin with other machines.

Supported by the Defense Advanced Research Projects Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Projects Agency or the U.S. Government.

Table of Contents

1 Introduction

This document explains how to use the Accent style Inter-Process Communication (IPC) facilities available under UNIX. [1] There are really three intertwined parts to these facilities. The first is only relevant if you are writing programs in UNIX PERQ Pascal (PP) and consists of an interface between PP and C. The second part is a mapping between Accent style IPC and UNIX IPC. The two IPC facilities are broadly similar since the Accent IPC was evolved from the UNIX IPC but there are many details which are not equivalent. The final part is an Accent style network server running under UNIX. This server is called the MsgServer under SPICE on the PERQs and is responsible for extending IPC over the network. The net result of these three parts is that you can write a program which uses Accent style IPC and will run on either a PERQ or a VAX with only minor changes. Furthermore it is simple to communicate between processes on any combination of the two machines.

The arrangement of this document does not quite follow the above threefold structure. Instead I devote one section to discussing how to use the IPC facilities from PP, another section to how to use the facilities from C and a third section to an explanation of the network services. A final section gives the current location of the relevant include, object and run files but these locations are liable to change and should be verified. Two appendices give examples of using the facilities from PP and C respectively.

I assume that you understand and are familiar with Accent IPC and, for instance, have programmed a PERQ to do inter-process communication. You should refer to the *Accent Kernel Interface Manual* for details of Accent IPC and to *ipc(2)* in the UNIX manual for details of UNIX IPC. Another *useful* document is my *SPICE System Programmers Guide*, available as */usr/rds/press/spicespg.press* on the SPICE VAX.

2 Perq Pascal

As mentioned in the introduction the interface from PP to the UNIX IPC facilities consists of an interface between PP and C and an interface between Accent and UNIX style IPC. The latter interface as well as the standard interface to UNIX IPC is written in C hence making the former interface necessary.

---

1 Unix is a Trademark of Bell Laboratories

2.1 Using the Interface

The functions that you can call from your PP program are specified in the module *accunix.p*. You should import this module into your program and it provides the bare necessities for Accent style IPC - if you need more then feel free to ask me. It is only a dummy module as the actual routines are written in C. The object code corresponding to these routines is in *accunix.o* and *caccunix.o*. You need both these object files - *accunix.o* does the parameter hacking from PP to C whilst *caccunix.o* is code that is common to both PP and C users. You will probably notice that the module *accunix.p* imports and exports the module *myacctype.p*. All that this latter module does is to define a generic pointer type before importing and exporting the most recent version of *AccentType.pas*, which should correspond with what you have on your PERQ. (The source of this module is */usr/accent/libpascal/src/#/accenttype.pas* on the SPICE VAX.)

Your compilation and link commands should look something like:

```
pp -c myprogram.p
pp myprogram.o accunix.o caccunix.o -lipc -o myoutput
```

The library *ipc* contains the actual C routines for doing UNIX IPC. It is important that *-lipc* occurs after the other object modules on the command line so that the loader looks for undefined routines in the *ipc* library after discovering them in the other object modules.p

2.2 UNIX and PERQs

Here are some of the differences that I have found between UNIX PP and the PP on the PERQs.

- If a routine is exported from a module then you should only declare it fully once in the *exports* section. In the *private* section of the module just put the name of the routine before you begin its body and leave out the parameter information.
- As far as I can tell the compiler invariably complains about repeated specifications if you import the same module more than once.
- As I mentioned above it is necessary to declare a generic pointer type as the UNIX PP compiler does not have it built in.
- The names of intrinsics vary wildly. Instead of *shrink* on the PERQ we have *narrow* on the VAX. Instead of *WordSize* we have *SizeOf*. *SizeOf* is much more restrictive, demanding as its parameter the actual variable for which you want the size. In addition it returns the number of bytes instead of the number of 16bit words - so beware.
- Long Octal constants are prefixed by *##* and not just one *#*.

3 C

3.1 The Interface

The only routines that I provide are *c_send* and *c_receive*. These provide the IPC translations as mentioned above. For all other IPC calls you should directly use the standard UNIX IPC calls, documented in ipc(2). These two routines are in the object file *caccunix.c*. Thus your compile and link commands should look something like:

```
cc -c myprogram.c
cc myprogram.o caccunix.o -lipc -o myoutput
```

The specifications of the two routines are as follows:

```
c_send (msghdr, maxwait, option)
        caddr_t msghdr;
        int maxwait, option;

c_receive (msghdr, maxwait, portopt, option)
        caddr_t msghdr;
        int maxwait, portopt, option;
```

The parameters are the same as for the Accent IPC routines *Send* and *Receive*. The first parameter should be a pointer to the message to be sent or available for containing a received message.

The include file *acctype.h* contains suitable definitions of Accent types and constants for you to construct your messages. If you find anything missing please feel free to ask me to add it.

3.2 Strings

If you are communicating with a PP program and you are passing strings between the two programs then you should be aware of the need to convert from C strings to PP strings. This should be done by the C program before sending the string. Macros for doing the conversion are available in the public include file *perq.h*. If you have a C string of 80 characters then the corresponding predefined PP string is *PerqString*. The conversion from C to PP should look like:

```
PerqString          myppstring;
char                * mycstring;
strcpy (myppsting.Chars, mycstring);  { Copies the characters upto a null. }
MakeCount (myppstring);               { Inserts the length as the first byte. }
```

4 Network Server

4.1 Introduction

The network server allows you to send IPC messages from machine to machine in a transparent manner. It provides a mapping between ports local to a machine and network ports. Thus if you have access to a port on another machine (either a VAX or a PERQ), you can send messages to that port provided you have a network server running on the machine. For historic reasons the network server on the PERQs is called the *MsgServer* whereas that on a VAX is still called the *NetServer*. The *NetServer* on a PERQ is the process that deals directly with the Ethernet on a packet level and should be called the *EtherServer*.

On most Vaxes in the Computer Science department there should be a network server running. You can check this by looking at the processes associated with user name *pupd*. There should be a process called *netserver*.

4.2 Using the NetServer

If you want to do remote IPC from a process under UNIX there are probably three phases.

1. Use the UNIX IPC name service local to your machine to locate the Accent style netserver. The name asserted by the netserver is *NAMESERVER*. This is a distinct difference from using the network server under Accent where, as part of your process' environment, you have access to the *nameserverport*.

   From PP you can use the routine in *accunix.p* called *locate*. Thus the call should look like:

   ```
   retval := locate ('NAMESERVER', nameserverport);
   ```

   where *nameserverport* is a **var** parameter.

   From C you should just use the *ipclocate* call of UNIX IPC:

   ```
   nameserverport = ipclocate ("NAMESERVER");
   ```

   If the call returns successfully then *nameserverport* will contain the local port giving you access to the name service facilities of the network server.

2. Use the name service facilities of the UNIX network server to look up the remote port with which you wish to communicate.

   From PP you can make a call to the function *lookup* provided in the module *MsgN*. This module is generated by Matchmaker and provides a procedural interface to the name services by a synchronous sending and receiving of appropriately constructed messages. Your program should look something like:

   ```
   imports MsgN    from MsgNUser;
   ```

```
retval := LookUp (nameserverport, 'REMOTENAME', remoteport);
```

You should not import *accunix* since *msgn* already imports and exports it. To gain access to these routines at link time you should, in addition to the object modules mentioned in the previous section, link with *msgnuser.o*.

From C there is a hand written equivalent of the Matchmaker generated PP interface. (Thanks go to Richard Goldschmidt for this.) The object code for this is contained in *cmsgnuser.o* and you should link your object code with this along with *caccunix.o* etc.. The call to *lookup* is all but identical to the PP call:

```
retval = lookup (nameserverport, "REMOTENAME", &remoteport);
```

except for the last parameter which, being a var parameter under PP, must be passed across as a pointer in this case.

In all cases, if the call to *LookUp* returns successfully, then *remoteport* will contain your local representation for the remote port. Under both languages you must initialise the Matchmaker interface to the network name services by calling the procedure *initmsgn* with a parameter of *NULLPORT*.

3. Now that you have obtained send rights on the remote port, just go ahead and send Accent style messages to it using the *accunix* subroutines. From now on the network server processes on your machine and on the remote machine are effectively transparent to you as a user, though of course there are quantative differences in send and receive times.

Once you have obtained send rights to a remote port you can send and receive rights on other ports using the normal IPC typing conventions. The network servers will do the appropriate mapping between remote and local ports.

5 Infelicities

5.1 IPC types

First some warnings about the IPC interface. You should be careful about what IPC type information you use. In particular I rely on the fact that when you give a type size in bits and the number objects then these figures define what you are giving me to the nearest word (where word is 16bits in PP and 32 bits in C). For instance you are better off declaring, though it is not mandatory, a *TYPEBOOLEAN* to have a *typesizeinbits* of 16 and not 1. Similarly a standard PP string of 80 characters should be declared to be 82 objects of *TYPECHAR* each of size 8 bits.

This brings up another point - it is recommended that you declare strings as being of *TYPECHAR* and not of *TYPESTRING* since the UNIX IPC type-size-in-bits field is only six bits wide and thus you can only use it for strings of upto 6 characters in length. I do try and cope with this problem for *TYPESTRING* but

is is not possible to deal with it for *TYPEUNSTRUCTURED*. Thus you must not make the *typesizeinbits* value for *TYPEUNSTRUCTURED* greater than or equal to 64. To get round this you can just increase the *numobjects* value until the product of the two quantities reaches the size of your object.

5.2 MsgSize

The *MsgSize* field in the message header must accurately reflect the amount of data you are trying to send or receive. I rely on this value to be correct when parsing the message body. On sending you should not leave any space between the end of the last item of data and the end of the message. Typically one uses the intrinsic *SizeOf* on a record or a structure to determine the size of the message.

5.3 Floating Point numbers

The format of floating point numbers differs between a PERQ and a VAX. Thus you should not try to send real quantities between the two machines. The only safe way of sending Floating Point numbers is to convert then into an implementation independent string of ASCII characters.

5.4 Size Limits

There is a limit within UNIX IPC of 4096 bytes on the size of the message. This implies that you should not make your Accent style message much bigger than 4000 bytes including any out of line data.

6 Location of Code

The optimised object modules *accunix.o*, *caccunix.o*, *msgnuser.o* and *cmsgnuser.o* are to be found in the directory */usr/rds/lib* on the SPICE VAX. Eventually this code should be put in something like */usr/local/lib.*

The C include file *acctype.h* and the PP imports modules *myacctype.p*, *accunix.p* and *msgnuser.p* are to be found in the directory */usr/rds/include* on the SPICE VAX. Eventually this code should be put in something like */usr/local/include.*

The run file for the *netserver* is to be found in the directory */usr/rds/bin* on the SPICE VAX.

Appendix A. PP Example

```
Program example;
{ Looks up a port checked in by the C program in Appendix B. }
{ Then sends a message to that port. }

imports msgn        from '/usr/rds/include/msgnuser.p';
    { N.B. msgnuser.p imports accunix.p so we must not import it here. }

var
    nameserverport  : port;
    sendtoport      : port;
    retval          : GeneralReturn;
    mymessage       : record
                        head     : Msg;
                        desc     : TypeType;
                        text     : string
                      end;

begin
    retval := NameNotCheckedIn;
    while retval <> success     { Locate the network nameserver. }
       do retval := locate ('NAMESERVER', nameserverport);

    InitMsgN (NULLPORT);        { Initialise the Matchmaker interface. }

    retval := NameNotCheckedIn;
    while retval <> success     { Locate the remote port. }
       do retval := LookUp (nameserverport, 'CPORT', sendtoport);

    with mymessage
    do begin
       Head.SimpleMsg           := true;
       Head.MsgSize             := SizeOf (mymessage);
       Head.MsgType             := NORMALMSG;
       Head.LocalPort           := NULLPORT;
       Head.RemotePort          := sendtoport;
       Head.ID                  := 42;

       desc.InLine              := true;
       desc.LongForm            := false;
       desc.DeAllocate          := false;
       desc.TypeName            := TypeChar;
       desc.TypeSizeInBits      := 8;
       desc.NumObjects          := 82;
       end;

    write ('Input: ');
    readln (mymessage.text);
    retval := send (mymessage.head, 0, WAIT);
    if retval = success
```

```
        then writeln ('Message sent.');
end.
```

Appendix B. C Example

```
/*
     Checks in a port called "CPORT".
     Waits to receive a message on that port from the PP program in Appendix A.
*/

#include <perq.h>
#include <acctype.h>

main ()
  {
    port            nameserverport;
    port            receiveport;
    generalreturn   retval;
    struct
        {
          struct msg       head;
          struct typetype  desc;
          PerqString       text;
         } mymessage;

    nameserverport = ipclocate ("NAMESERVER");
                                        /* Locate network name server. */

    receiveport = ipcallocateport (5);  /* Create a port. */

    initmsgn (NULLPORT);                /* Initialise the Matchmaker interface. */

    retval = checkin (nameserverport, "CPORT", NULLPORT, receiveport);
                                        /* Check in the Port. */

    /* Initialise the message. */
    mymessage.head.msgsize    = sizeof (mymessage);
    mymessage.head.localport  = receiveport;

    /* Wait to receive the message. */
    retval = c_receive (&mymessage, 0, LOCALPT, RECEIVEIT);

    if (retval == ASUCCESS)
      {
        MakeAsciz (mymessage.text);     /* Convert PP string to C string. */
        printf ("Message received: %s\n", mymessage.text.Chars);
      }
  }
```

Appendix C. Makefile Example

The makefile for the above two programs should, assuming you are using the Spice Vax, look like:

```
CFLAGS = "-I/usr/rds/include"
# First the Pascal program
ppexam:         ppexam.o
                pp ppexam.o /usr/rds/lib/msgnuser.o /usr/rds/lib/accunix.o \
                        /usr/rds/lib/caccunix.o -lipc -o ppexam
ppexam.o:       ppexam.p
                pp -c ppexam.p
# Now the C program
cexam:          cexam.o
                cc cexam.o /usr/rds/lib/cmsgnuser.o /usr/rds/lib/caccunix.o -lipc -o cexam
```